МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА імені О. М. БЕКЕТОВА**

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до виконання лабораторних та практичних робіт
із навчальної дисципліни

«ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ»

*(для студентів 1–2 курсів усіх форм навчання спеціальності
122 – Комп'ютерні науки та інформаційні технології галузі знань
12 – Інформаційні технології)*

**Харків
ХНУМГ ім. О. М. Бекетова
2018**

Методичні рекомендації до виконання лабораторних та практичних робіт із навчальної дисципліни «Організація баз даних та знань (для студентів 1–2 курсів усіх форм навчання спеціальності 122 – Комп'ютерні науки та інформаційні технології галузі знань 12 – Інформаційні технології) / Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова ; уклад. : О. Б. Костенко, І. О. Гавриленко. – Харків : ХНУМГ ім. О. М. Бекетова, 2018. – 19 с.

Укладачі: канд. фіз.-мат. наук. О. Б. Костенко,
ас. І. О. Гавриленко

Рецензент
М. Ю. Карпенко, кандидат технічних наук, доцент Харківського національного університету міського господарства імені О. М. Бекетова

Рекомендовано кафедрою прикладної математики і інформаційних технологій, протокол № 2 від 06.09.2017.

ЗМІСТ

1 МЕТА ЛАБОРАТОРНИХ ТА ПРАКТИЧНИХ ЗАНЯТЬ

Мета роботи: сформувати у студентів навички практичного застосування існуючих систем управління базами даних; вживання ефективних моделей забезпечення даних на основі вивчення предметної області, методів аналізу, пошуку та використання існуючих систем управління базами даних; знайомство з існуючими системами управління базами даних реляційного типу; забезпечення теоретичної та інженерної підготовки фахівців у галузі проектування та використання систем управління базами даних.

Згідно з вимогами освітньо-професійної програми при вивченні навчальної дисципліни студенти повинні оволодіти:

– основним поняттями баз даних;

– поняттями архітектури та концепції систем управління базами даних;

– методами побудови структур і схем зберігання даних;

– способами проектування баз даних на концептуальному і логічному рівнях, а також при їх фізичній реалізації.

В процесі виконання лабораторних та практичних занять студенти повинні навчитися визначати предметну область бази даних, визначати об’єкти, що підлягають представленню в базі даних, будувати формалізований опис об’єктів, визначати первинні та зовнішні ключі, а також розробляти SQL-запити для зміни структури таблиць бази даних (включення нових полів, вилучення полів таблиць, зміни опису полів, зміни обмежень). Для зміни структури таблиць використовуються директиви мови SQL, які студенти повіинні засвоїти в процесі виконання роботи.

2 ВИМОГИ ДО ЗВІТУ З ЛАБОРАТОРНОЇ ТА ПРАКТИЧНОЇ РОБОТИ

Звіт щодо роботи повинен бути представлений в електронній та друкованій версіях.

Електронна версія (на магнітному носії) містить:

1. Файл звіту в форматі документу Word.
2. Базу даних (файл Access) за Вашим варіантом.

Друкована версія містить:

1. Друковану копію файлу звіту в форматі документу Word.
2. Файл звіту: «Звіт_прізвище». Наприклад, «Звіт_Костенко».

Зміст файлу звіту:

1. Титульний аркуш (зразок оформлення титульного аркушу наведено в додатку А).
2. Завдання на роботу (див. нижче).
3. Конспект ресурсу (п. 2 завдання).
4. Побудова моделі (п. 3 завдання).
5. Роздруківка таблиць бази даних з внесеними даними.
6. Тексти SQL-запитів до бази даних і результати цих запитів в табличній формі.
7. Висновок про можливості інформатизації обраної предметної області.

Загальний обсяг файлу звіту 7–10 сторінок.

Порядок здачі роботи:

1. Пред’явлення файлів, що містять виконані завдання.
2. Співбесіда на основі друкованої копії звіту і пред’явлених файлів.

Примітка 1. Можлива поетапна здача роботи.

Примітка 2. Робота є допуском до іспиту.

3 ЗАВДАННЯ ДО ЛАБОРАТОРНИХ ТА ПРАКТИЧНИХ РОБІТ

1. Узгодити з викладачем предметну область для створення проекту інформаційної системи.

2. Опрацювати ресурс «Основи проектування баз даних».

2.1. Скласти детальний конспект глави 2 зазначеного ресурсу. Помістити туди всі визначення: «сутність», «асоціація», «ключ» та ін.

2.2. Законспектувати главу 3 зазначеного ресурсу. Поняття для усвідомлення: «відношення», «операції з відношеннями», «атрибут», «кортеж», «домен», «ступінь відношення». Розуміння принципів побудови баз даних (розділ 3.2 ресурсу «Основи проектування баз даних»).

2.3. Законспектувати главу 4 зазначеного ресурсу. Звернути увагу на наступні аспекти:

– «нормалізація» та її цілі;
– «універсальне відношення»;
– необхідність розбиття «універсального відношення»;
– «нормальні форми»;
– процедура «нормалізації»;
– мова інфологічного моделювання «Таблиці – зв’язки».

3. Побудувати інфологічну модель предметної області за Вашим варіантом. Для цього:

– виділити «стержневі сутності» та їх «атрибути» (описово);
– побудувати зв’язки («асоціації», «характеристики») і їх «атрибути» (описово);
– визначити «первинні» та «зовнішні ключі» (описово);
– записати модель на мові «ER-діаграм».

Конспект (п. 2) і побудову інфологічної моделі (п. 3) виконати в файлі Word і помістити в звіт.

4. Створити таблиці бази даних, які відповідають інфологічної моделі, і внести в них по сім записів. На схемі даних показати зв’язок між об’єктами моделі.

5. Створити запити до бази даних в формі SQL, які б розкривали сутність інформаційної системи і відповідали на більшість питань користувача Вашої інформаційної системи. Тексти запитів і результат їх роботи помістити в звіт.

6. Підготувати і роздрукувати файл звіту.

Примітка 1. Конспект повинен містити приклади з створеної інформаційної системи.

Примітка 2. Ім’я бази даних – ім’я інформаційної системи «Ваше_прізвище», наприклад, «Ресторан_Костенко».

4 МЕТОДИКА ВИКОНАННЯ РОБІТ

4.1 Варіанти предметних областей для створення проекту інформаційної системи

Предметною областю називається частина реального світу, що представляє інтерес для даного дослідження. В автоматизованих інформаційних системах відображення предметної області представлено моделями даних декількох рівнів. Кількість рівнів моделей буде залежати від особливостей системи управління базами даних (СУБД). Можливі варіанти предметних областей для створення проекту інформаційної системи:

1. Продуктовий кіоск.
2. Хіти сезону та виконавці.
3. Виконавці популярної музики і альбоми.
4. Розклад літаків.
5. Розклад поїздів.
6. Розклад міжміських автобусів.
7. Відділ кадрів.
8. Готель.
9. Гуртожиток.
10. Комп’ютерний магазин.
11. Склад.
12. Бібліотека.
13. Кулінарний довідник.
14. Канцелярія.
15. Лікарня.
16. Майстерня з ремонту взуття.
17. Станція технічного обслуговування автомобілів.
18. Студенти і захоплення.
19. Студенти і спортивні секції.
20. Викладачі та предмети.
21. Кафедри і викладачі.
22. Студенти і улюблені страви.
23. Студенти і оператори мобільного зв’язку.
24. Замовлення таксі.

4.2 Рекомендації до інфологічного моделювання предметної області

Найменша одиниця даних реляційної моделі – це окреме атомарне (що не розкладається) для даної моделі значення даних. Так, в одній предметній області прізвище, ім'я та по батькові можуть розглядатися як єдине значення, а в іншій – як три різних значення.

Доменом називається безліч атомарних значень одного і того ж типу. Якщо значення двох атрибутів беруться з одного і того ж домена, то, ймовірно, мають сенс порівняння, які використовують ці два атрибути. Якщо ж значення двох атрибутів беруться з різних доменів, то їх порівняння, ймовірно, позбавлене сенсу.

Відношення на доменах D_1, D_2, ..., D_n складається із заголовка і тіла.

Заголовок складається з такої фіксованої множини атрибутів A_1, A_2, ..., A_n, що існує взаємно однозначна відповідність між цими атрибутами A_i і доменами D_i (i = 1, 2, ... , n), що їх визначають.

Тіло складається із змінюваної в часі множини *кортежів*, де кожен кортеж складається в свою чергу з безлічі пар атрибут-значень (A_i : V_i), (i = 1, 2, ..., n), по одній такій парі для кожного атрибута A_i в заголовку. Для будь-якої заданої пари атрибут-значень (A_i : V_i) V_i є значенням з єдиного домену D_i, який пов'язаний з атрибутом A_i.

Ступінь відношення – це число його атрибутів. Відношення ступеня один називають унарним, ступеня два – бінарним, ступеня три – тринарним, ..., ступеня n – n-арним. *Кардинальне число* або *потужність відношення* – це число його кортежів. Кардинальне число відношення змінюється в часі на відміну від його ступеня.

Оскільки відношення – це множина, а множина за визначенням не містить співпадаючих елементів, то ніякі два кортежі відношення не можуть бути дублікатами один одного в будь-який довільно-заданий момент часу.

Нехай R – відношення з атрибутами A_1, A_2, ..., A_n. Кажуть, що множина атрибутів K = (A_i, A_j, ..., A_k) відношення R є *можливим ключем* R тоді і тільки тоді, коли задовольняються дві незалежні від часу умови.

1. Унікальність: у довільний заданий момент часу ніякі два різних кортежа R не мають одного і того ж значення для A_i, A_j, ..., A_k.

2. Мінімальність: жоден з атрибутів A_i, A_j, ..., A_k не може бути виключений з K без порушення унікальності.

Кожне відношення володіє хоча б одним можливим ключем, оскільки комбінація всіх його атрибутів задовольняє умові унікальності. Один з можливих ключей, який обраний довільним образом, приймається за його

первинний ключ. Інші можливі ключі, якщо вони є, мають назву *альтернативні ключі*.

Вищезазначені та деякі інші математичні поняття явилися теоретичною базою для створення реляційних СУБД, розробки відповідних мовних засобів і програмних систем, що забезпечують їх високу продуктивність, і створення основ теорії проектування баз даних (БД). Однак для масового користувача реляційних СУБД можна з успіхом використовувати неформальні еквіваленти цих понять:

Відношення – Таблиця (іноді Файл),

Кортеж – Рядок (Запис),

Атрибут – Стовпець (Поле).

При цьому приймається, що «запис» означає «екземпляр запису», а «поле» означає «ім'я і тип поля».

Реляційна база даних – це сукупність відносин, що містять всю інформацію, яка повинна зберігатися в базі даних. Однак користувачі можуть сприймати таку базу даних як сукупність таблиць. Існують наступні правила.

1. Кожна таблиця складається з однотипних рядків (записів) і має унікальне ім'я.

2. Рядки (записи) мають фіксоване число полів (стовпців) і значень (множинні поля і повторювані групи неприпустимі). Інакше кажучи, в кожній позиції таблиці на перетині рядка і стовпця завжди є в точності одне значення або нічого.

3. Рядки таблиці обов'язково відрізняються один від одного хоча б єдиним значенням, що дозволяє однозначно ідентифікувати будь-який рядок такої таблиці.

4. Стовпцям таблиці однозначно присвоюються імена і в кожному з них розміщуються однорідні значення даних (дати, прізвища, цілі числа або грошові суми).

5. Повний інформаційний зміст бази даних представляється у вигляді явних значень даних і такий метод подання є єдиним. Зокрема, не існує будь-яких спеціальних «зв'язків» або покажчиків, що з'єднують одну таблицю з іншою.

6. При виконанні операцій з таблицею її рядки і стовпці можна обробляти в будь-якому порядку без відношення до їх інформаційного змісту. Цьому сприяє наявність імен таблиць та їх стовпців, а також можливість виділення будь-якого їх рядка або будь-якого набору рядків із зазначеними ознаками.

4.3 Рекомендації до проведення процедури «нормалізації»

Нормалізація – це розбиття таблиці на дві або більше, що володіють кращими властивостями при включенні, зміні і видаленні даних. Остаточна мета нормалізації зводиться до отримання такого проекту бази даних, в якому кожен факт з'являється лише в одному місці, тобто виключена надмірність інформації. Це робиться не стільки з метою економії пам'яті, скільки для виключення можливої суперечливості збережених даних.

Кожна таблиця в реляційній БД задовольняє умові, відповідно до якої в позиції на перетині кожного рядка і стовпчика таблиці завжди знаходиться єдине значення, і ніколи не може бути безліч таких значень. Будь-яка таблиця, яка задовольняє цій умові, називається *нормалізованою*. Фактично, ненормалізовані таблиці, тобто таблиці, що містять повторювані групи, навіть не допускаються в реляційної БД.

Будь-яка нормалізована таблиця автоматично вважається таблицею в першій нормальній формі (1НФ). Однак на практиці термін «нормалізована» часто використовується в більш вузькому сенсі – «повністю нормалізована», який означає, що в проекті не порушуються ніякі принципи нормалізації.

Таблиця знаходиться в *першій нормальній формі (1НФ)* тоді і тільки тоді, коли жодна з її рядків не містить в будь-якому своєму полі більше одного значення і жодне з її ключових полів не порожньо.

Таблиця знаходиться в *другій нормальній формі (2НФ)*, якщо вона задовольняє визначенню 1НФ і всі її поля, що не входять в первинний ключ, пов'язані повною функціональною залежністю з первинним ключем.

Для спрощення нормалізації таблиць доцільно використовувати наступну рекомендацію. При проведенні нормалізації таблиць, в які введені цифрові або інші замінники складових і (або) текстових первинних і зовнішніх ключів, слід хоча б подумки підміняти їх на вихідні ключі, а після закінчення нормалізації знову відновлювати.

Таблиця знаходиться в *третій нормальній формі (3НФ),* якщо вона задовольняє визначенню 2НФ і не одне з її неключових полів не залежить функціонально від будь-якого іншого неключового поля.

Таблиця знаходиться в *нормальній формі Бойса-Кодда (НФБК),* якщо і тільки якщо будь-яка функціональна залежність між його полями зводиться до повної функціональної залежності від можливого ключа.

У наступних нормальних формах (4НФ і 5НФ) враховуються не тільки функціональні, але і багатозначні залежності між полями таблиці. Для їх опису познайомимося з поняттям повної декомпозиції таблиці.

Повною декомпозицією таблиці називають таку сукупність довільного числа її проекцій, з'єднання яких повністю збігається із вмістом таблиці.

Таблиця міститься в п'ятій нормальній формі (5НФ) тоді і тільки тоді, коли в кожній її повної декомпозиції всі проекції містять можливий ключ. Таблиця, яка не має жодної повної декомпозиції, також знаходиться в 5НФ.

Четверта нормальна форма (4НФ) є окремим випадком 5НФ, коли повна декомпозиція повинна бути з'єднанням рівно двох проекцій. Дуже не просто підібрати реальну таблицю, що може бути надана в 4НФ, але не була б в 5НФ.

Теорія нормалізації ґрунтується на наявності тієї чи іншої залежності між полями таблиці. Визначено два види таких залежностей: функціональні і багатозначні.

Функціональна залежність. Поле B таблиці функціонально залежить від поля A тієї ж таблиці в тому і тільки в тому випадку, коли в будь-який заданий момент часу для кожного з різних значень поля A обов'язково існує тільки одне з різних значень поля B. Відзначимо, що тут допускається, що поля A та B можуть бути складовими.

Повна функціональна залежність. Поле B знаходиться в повній функціональної залежності від складеного поля A, якщо воно функціонально залежить від A і не залежить функціонально від будь-якої підмножини поля A.

Багатозначна залежність. Поле A багатозначно визначає поле B тій же таблиці, якщо для кожного значення поля A існує певна безліч відповідних значень B.

Інше визначення «нормалізації» – це процес послідовної заміни таблиці її повними декомпозиціями до тих пір, поки всі вони не будуть знаходитися в 5НФ. На практиці ж досить привести таблиці до НФБК і з великою гарантією вважати, що вони перебувають в 5НФ.

Процедура приведення таблиць до НФБК ґрунтується на тому, що єдиними функціональними залежностями в будь-якій таблиці повинні бути залежності виду K → F, де K – первинний ключ, а F – деяке інше поле. Це випливає з визначення первинного ключа таблиці, відповідно до якого K → F завжди має місце для всіх полів цієї таблиці. Мета нормалізації полягає саме в тому, щоб позбутися від усіх цих «інших» функціональних залежностей, тобто таких, які мають інший вигляд, ніж K → F.

Якщо скористатися рекомендацією, наведеною вище, і підмінити на час нормалізації коди *первинних (зовнішніх)* ключів на *вихідні ключі*, то, по суті, слід розглянути лише два випадки.

1. Таблиця має складовою первинний ключ виду ($К_1$, $К_2$) і включає також поле F, яке функціонально залежить від частини цього ключа, наприклад, від $К_2$, тільки не від повного ключа. В цьому випадку рекомендується сформувати іншу таблицю, яка містить $К_2$ і F (первинний ключ – $К_2$), і видалити F з початкової таблиці:

замінити T ($К_1$, $К_2$, F), первинний ключ ($К_1$, $К_2$), $Ф_3$ $К_2 \rightarrow F$

на $Т_1$ ($К_1$, $К_2$), первинний ключ ($К_1$, $К_2$),

і $Т_2$ ($К_2$, F), первинний ключ $К_2$.

2. Таблиця має первинний (можливий) ключ К, поле F_1, яке не є можливим ключом, що, звичайно, функціонально залежить від К, і інше неключове поле F_2, яке функціонально залежить від F_1. Рішення те ж саме, що і раніше – формується інша таблиця, яка містить F_1 і F_2, з первинним ключем F_1, і F_2 видаляється з початкової таблиці:

замінити T (К, F_1, F_2), первинний ключ К, $Ф_3$ $F_1 \rightarrow F_2$

на $Т_1$ (К, F_1), первинний ключ К,

і $Т_2$ (F_1, F2), первинний ключ F_1.

Для будь-якої заданої таблиці, повторюючи застосування двох розглянутих правил, майже у всіх практичних ситуаціях можна отримати в кінцевому рахунку безліч таблиць, які знаходяться в «остаточній» нормальній формі і, таким чином, не містять будь-яких функціональних залежностей виду, відмінного від К→F.

Для виконання цих операцій необхідно спочатку мати в якості вхідних даних будь-які «великі» таблиці (наприклад, універсальні відносини).

4.4 SQL-запити

Запит являє собою якусь команду, яка звертається до БД і повідомляє їй, щоб вона відобразила певну інформацію з таблиць в пам’ять. Ця інформація зазвичай виводиться безпосередньо на екран комп’ютера, термінал, надсилається принтеру, зберігається в файлі або служить вихідними даними для іншої команди або запиту.

Всі SQL-запити складаються з одиночної команди SELECT з досить простою структурою, проте шляхом її використання можна виконати складну обробку даних. У найпростішій формі команда SELECT звертається до БД, щоб отримати інформацію з таблиці. Наприклад, щоб вивести таблицю студентів, слід дати наступний запит:

```
SELECT SNUM, SFAM, SIMA, SOTCH, STIP FROM STUDENTS;
```

У цій команді SELECT – ключове слово, яке повідомляє БД, що ця команда є запитом, тобто всі запити починаються цим словом.

SNUM, SFAM, SIMA, SOTCH, STIP – список полів з таблиці, які вибираються запитом. Поля, не перераховані тут, не будуть включені в висновок команди. Запит не вплине на інформацію в таблицях; він тільки показує дані. FROM STUDENTS – ключове слово, подібно SELECT, яке повинно бути представлено в кожному запиті. Воно супроводжується

пропуском і потім ім’ям таблиці використовуваної в якості джерела інформації. В даному випадку – це таблиця студентів STUDENTS.

Крапка з комою («;») використовується у всіх інтерактивних командах SQL для повідомлення БД, що команда заповнена і готова виконатися, а в деяких системах похила риса в рядку є індикатором кінця команди. Очевидно, запит такого характеру не обов’язково буде впорядковувати висновок будь-яким зазначеним способом. Та ж сама команда, виконана з тими ж самими даними, але в різний час не зможе вивести результат в однаковому порядку. Зазвичай рядки виявляються в тому порядку, в якому вони знайдені в таблиці, а оскільки він довільний, то зовсім не обов’язково буде зберігатися той порядок, в якому дані вводилися або зберігалися.

Якщо необхідно отримати кожне поле таблиці, є необов’язкове скорочення у вигляді символу «зірочка» (*), яке можна використовувати для виведення повного списку полів наступним чином:

SELECT * FROM STUDENTS;

що призведе до того ж результату, що і попередня команда. У загальному випадку запит починається з ключового слова SELECT, супроводжуваного пробілом. Після цього повинен слідувати список розділених комами імен полів, які необхідно вивести.

Ключове слово FROM, наступне далі, супроводжується пропуском і ім’ям таблиці, запит до якої робиться. Крапка з комою повинна використовуватися для того, щоб закінчити запит і вказати що команда готова до виконання. Команда SELECT здатна витягти строго певну інформацію з таблиці. Наприклад, при необхідності виведення тільки певних полів таблиці, просто зі списку виключаються непотрібні поля.

Цей спосіб дозволяє працювати з таблицями, які мають велику кількість полів, що містять дані, які не потрібні в даний момент користувачеві.

При роботі з даними дуже часто виникає потреба у видаленні надлишкових даних. Це реалізується використанням DISTINCT – аргументом, який забезпечує можливість усувати повторювані значення з пропозиції SELECT. Припустимо, що необхідно дізнатися, які студенти в даний час здавали навчальні предмети, причому не потрібне уточнення отриманої оцінки і здається предмета. Запит

SELECT SNUM FROM USP;

надає наступний висновок, проте в ньому є записи-дублікати:

SNUM
3412
3413
3414

3412

3416

Для отримання списку результатів без дублікатів в даному випадку доцільно скористатися наступним:

SELECT DISTINCT SNUM FROM USP;

в результаті чого буде отримано:

SNUM

3412

3413

3414

3416

Слід мати на увазі, що DISTINCT може вказуватися тільки один раз в даній пропозиції SELECT. Якщо пропозиція вибирає численні поля, DISTINCT опускає записи, де всі вибрані поля ідентичні. Якщо замість DISTINCT вказати ALL, то це буде мати протилежний ефект і дублювання рядків виводу збережеться.

Для використання умов пошуку для відбору рядків слід користуватися наступними командами. WHERE – пропозиція команди SELECT, яка дозволяє встановлювати предикати, умова яких може бути або вірнім або невірнім для будь-якого запису таблиці. Команда витягує тільки ті записи з таблиці, для якої таке тверджень істинне. Припустімо, що необхідно вибрати прізвища и розміри стипендії студентів, при цьому цікавлять тільки такі, які отримують стипендію в розмірі 25.50. Такий запит буде мати вигляд:

SELECT SFAM, STIP

FROM STUDENTS WHERE

STIP=25.50;

Часто в предикатах потрібно не тільки оцінювати рівність оператора як істинного або хибного, а й здійснювати інші види зв’язків. Це реалізується за допомогою булевих операторів і знаків відносини, причому предикат може містити необмежену кількість умов. В цілому, реляційний оператор – це математичний символ, який вказує на певний тип порівняння між двома значеннями, при цьому SQL має наступний їх набор:

= рівний чогось;

> більш ніж;

< менш ніж;

>= більш ніж або дорівнює;

<= менш ніж або дорівнює;

<> не дорівнює.

Ці оператори мають стандартні значення для числових даних, а для символьних їх визначення залежить від кодів ASCII символів – вони слідують в алфавітному порядку, причому великі літери мають менший код, ніж малі, тому, наприклад, «Z» < «a». Припустимо, що необхідно вивести список студентів, які отримують стипендію, тобто для яких STIP > 0. Для цього скористаємося наступним запитом:

```
SELECT *
FROM STUDENTS
WHERE STIP > 0;
```

Стандартними булевими операторами, які використовуються в SQL, є «AND», «OR» і «NOT». Вони працюють наступним чином.

– «AND» використовує два операнда в формі «A AND B» і оцінює їх по відношенню до істини: чи вірні вони обидва;

– «OR» використовує два операнда в формі «A OR B» і оцінює на істинність: чи вірний один з них;

– «NOT» використовує один операнд в формі «NOT A» і замінює його значення з «ІСТИНА» на «НЕІСТИНА», або навпаки.

Умова «NOT» може використовуватися для інвертування логічних значень. Наприклад, для виведення інформації про студентів, у яких оцінки не є 3, можна скористатися наступним запитом:

```
SELECT * FROM USP WHERE NOT (OCENKA = 3);
```

Слід мати на увазі, що булевский оператор розміщується перед реляційним оператором, на який він діє, а при необхідності розширення дії використовуються дужки. Наприклад, для виведення інформації про студентів, у яких оцінки не є 3 і в той же час з навчального предмета з кодом, не рівним 2005, можна скористатися таким запитом:

```
SELECT * FROM USP WHERE NOT {OCENKA = 3 AND PNUM = 2005);
```

У цьому випадку SQL розуміє круглі дужки як підтвердження, що все всередині них буде оцінюватися першим і оброблятися як єдине вираження за допомогою того оператора, який знаходиться зовні. У реченні SELECT на додаток до традиційних реляційних і булевих операторів, розглянутим вище, можуть бути використані спеціальні оператори «IN», «BETWEEN», «LIKE» і «IS NULL». Оператор «IN» визначає набір значень, в який дане значення має бути включено. Наприклад, якщо відповідно до навчальної БД виникає необхідність в поданні інформації про всіх студентів, ім’я яких «Анатолій» або «Володимир», потрібно виконати такий запит:

```
SELECT *
FROM STUDENTS
WHERE SIMA = ‘Анатолій’ OR
```

SIMA = ‘Володимир’;

Однак є і більш простий спосіб отримати ту ж інформацію за допомогою запиту:

```
SELECT *
FROM STUDENTS
WHERE SIMA IN (‘Анатолій’, ‘Володимир’);
```

Звідси ясно, що «IN» визначає набір значень за допомогою списку, укладеного в круглі дужки з роздільниками у вигляді ком. Він перевіряє різні значення вказаного поля, намагаючись знайти збіг із значеннями з набору. Якщо це трапляється, то предикат вірний. Якщо набір містить числові значення, а не символьні, то використовуйте одиничні лапки опускаються. Прикладом такого запиту може служити пошук всіх студентів, які мають стипендію 17.00 і 25.50:

```
SELECT *
FROM STUDENTS
WHERE STIP IN (17.00, 25.50);
```

Оператор «BETWEEN» дещо схожий на «IN», але на відміну від визначення з набору, «BETWEEN» визначає діапазон значень, в який повинні уміщатися шукані значення, що і робить предикат вірним. Структура оператора «BETWEEN» наступна: вводиться початкове значення, ключове слово «AND» і кінцеве значення. На відміну від «IN», «BETWEEN» розрізняє порядок значень, отже, перше з них в пропозиції повинне бути першим по алфавітному або числовому порядку. Наступний приклад буде відбирати з таблиці успішності номера і оцінки всіх студентів, оцінки яких укладені між 3 і 5:

```
SELECT SNUM, OCENKA FROM USP WHERE
OCENKA BETWEEN 3 AND 5;
```

Оператор «LIKE» застосовний тільки до полів типу «CHAR» або «VARCHAR», в яких він шукає підрядки, тобто він шукає символи і перевіряє, чи збігаються вони з умовою. Як умова оператор використовує групові символи – спеціальні символи, які відповідають будь-чому. Існує два типи групових символів, використовуваних з «LIKE»:

– символ підкреслення заміщує будь-який одиночний символ, наприклад, «М_Л» буде відповідати словами «МОЛ» або «МЕЛ», але не буде відповідати «МЕТАЛ»;

– символ відсотка заміщує послідовність будь-якого чис ла символів, в тому числі нульової довжини. Наприклад, «% М% Л» буде відповідати словами «МЕЛ» або «ПОМОЛ», але не відповідає «МОЛОКО».

Як приклад, знайдемо всіх викладачів, прізвища яких починаються з літери «К»:

```
SELECT TFAM, TIMA, TOTCH
```

FROM TEACHERS WHERE

TFAM LIKE ‘K%’;

Для отримання підсумкових даних використовуються агрегатні функції. За допомогою їх запити можуть виконувати узагальнену групову обробку значень полів. У SQL допускаються наступні агрегатні функції.

«COUNT» – проводить підрахунок кількості рядків або «НЕ-NULL значень полів, які вибрав запит.

«SUM» – розраховує арифметичну суму всіх обраних значень даного поля.

«AVG» – виробляє усереднення всіх обраних значень даного поля.

«MAX» – знаходить і повертає найбільше з усіх обраних значень даного поля.

«MIN» – знаходить і повертає найменше з усіх обраних значень даного поля.

Варто мати на увазі, що з «SUM» і «AVG» використовуються тільки числові поля, а з «COUNT», «MAX», і «MIN» можуть використовуватися числові або символьні поля. Коли функції записуються з символьними полями. «MAX» і «MIN» використовують їх в еквівалент ASCII, відповідно до якого вибирається максимальне або мінімальне значення. Щоб знайти суму всієї виплаченої стипендії в таблиці з даними про студентів, варто виконати такий запит:

SELECT SUM (STIP)

FROM STUDENTS;

результат якого складається з одного значення 68.00.

Команда «GROUP BY» може застосовуватися з агрегатними функціями незалежно від серій груп, які визначаються за допомогою значення поля в цілому. У цьому випадку кожна група складається з усіх рядків з тим же самим значенням поля «SNUM», і «MIN» функція застосовується окремо для кожної такої групи. Значення поля, до якого застосовується «GROUP BY», має тільки одне значення на групу виводу, так само як це робить агрегатна функція. Тому в результаті і з’являється сумісність, яка дозволяє агрегатним функціям і полях об’єднатися. Також допускається використання «GROUP BY» одночасно з декількома полями. Для прикладу створимо вибір найменшого значення оцінки за кожен день. При цьому запит буде наступний:

SELECT SNUM, UDATE, MIN (OCENKA)

FROM USP

GROUP BY SNUM, UDATE;

При багатотабличному запиті таблиці, представлені у вигляді списку в «FROM», відокремлюються один від одного комами. Предикат запиту може

посилатися до будь-якому стовпцю будь-якої пов’язаної таблиці і, отже, може використовуватися для зв’язку між ними. Зазвичай предикат порівнює значення в стовпцях різних таблиць, щоб визначити, чи задовольняє «WHERE» встановленій умові.

Припустимо необхідно поставити у відповідність викладачу навчальні предмети, які він веде. Фактично SQL доведеться вибирати з таблиці викладачів відповідний йому код і, переглядаючи таблицю предметів, здійснювати пошук відповідного коду. Це можна реалізувати наступним запитом:

```
SELECT TEACHERS.TFAM, PREDMET.PNAME
FROM TEACHERS, PREDMET WHERE
TEACHERS.TNUM = PREDMET.TNUM;
```

Оскільки поле TNUM є і в таблиці викладачів, і в таблиці предметів, то імена таблиць повинні використовуватися як префікси, хоча це необхідно тільки в тому випадку, коли два або більше полів мають одне і те ж ім’я. Однак в будь-якому випадку включати ім’я таблиці в запити з об’єднанням зручно для кращого розуміння і несуперечності отриманих результатів. При виконанні багатотабличного запиту SQL досліджує кожну комбінацію рядків двох або більше можливих таблиць і перевіряє ці комбінації по їх предикатам. Якщо комбінація виробляє таке значення, яке робить предикат вірним, то значення буде вибрано для виводу.

Об’єднання в багатотабличних запитах, які використовують предикати, засновані на рівності, називаються об’єднаннями за рівністю. Об’єднання за рівністю – це найбільш загальний вигляд об’єднання, однак існують і інші види об’єднань. Фактично можна використовувати будь-який з реляційних операторів. Прикладом іншого виду об’єднання може служити наступний запит:

```
SELECT TEACHERS. TFAM, PREDMET . PNAME
FROM TEACHERS, PREDMET WHERE
TEACHERS. TFAM < PREDMET. PNAME
AND TEACHERS.TFAM BETWEEN ‘K’ AND ‘C’ ;
```

За допомогою цього запиту виводиться інформація про викладачів, прізвище яких за алфавітом знаходиться раніше, ніж найменування навчального предмета з таблиці «PREDMET», при цьому вибираються викладачі з прізвищем, яке знаходиться між буквами «К» та «С».

ДОДАТОК А

Зразок оформлення титульного аркуша

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА імені О. М. БЕКЕТОВА

Кафедра прикладної математики і інформаційних технологій

ЗВІТ

з виконання роботи
з дисципліни
«Організація баз даних та знань»

Виконав:
студент(ка) __ курсу
фак-ту менеджменту
групи _____
ПІБ (повністю)

Перевірив:

Харків
ХНУМГ ім. О. М. Бекетова
2018

Виробничо-практичне видання

Методичні рекомендації
до виконання лабораторних та практичних робіт
із навчальної дисципліни

«ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ»

(для студентів 1–2 курсів усіх форм навчання спеціальності 122 – Комп'ютерні науки та інформаційні технологі галузі знань 12 – Інформаційні технології)

Укладачі: **КОСТЕНКО** Олександр Борисович,
ГАВРИЛЕНКО Ірина Олександрівна

Відповідальний за випуск *О. Б. Костенко*

За авторською редакцією

Комп'ютерне верстання *І. В. Волосожарова*

План 2018, поз. 361 М

Підп. до друку 03.05.2018. Формат 60 × 84/16.
Друк на ризографі. Ум. друк. арк. 1,0.
Тираж 50 пр. Зам. № .

Видавець і виготовлювач:
Харківський національний університет
міського господарства імені О. М. Бекетова,
вул. Маршала Бажанова, 17, Харків, 61002.
Електронна адреса: rectorat@kname.edu.ua
Свідоцтво суб'єкта видавничої справи:
ДК № 5328 від 11.04.2017.